# SOFTWARE SYSTEM SAFETY GUIDE

LEONARD L. RUSSO
US ARMY COMMUNICATIONS AND
ELECTRONICS COMMAND

MAY 92

## SUMMARY

Because software is unsuited to traditional hardware-oriented hazard analysis techniques, System Safety Engineers must ensure that Software Safety requirements are part of the specification documents. Selected software hazard analyses should be used during the design phases to detect software deficiencies and assure that adequate safety features are designed into the software. This Software Safety Guide is provided to assist the System Safety Engineer in developing and/or managing a Software Safety Program and provide insight into the safety requirements for the design of safety critical software. The Safety engineer faces new challenges when integrating Software Safety into the total system safety effort. To assist the System Safety Engineer with implementation, this guide will:

* Define Software and Software Safety terms.

* Explain the use of the Software Safety Matrix.

* Discuss the implementation of the Software Safety Program Plan (SWSPP).

* Discuss Software Safety requirements in the Statement of Work (SOW).

* Discuss software documentation and configuration management.

* Discuss previously developed or NDI software.

* Discuss Software Safety analysis/guidelines.

* Provide an example of a Software Safety Design Verification Checklist.

# FOREWORD

This Technical Report was prepared by the U.S Army Materiel Command (AMC) Action Committee for System Safety (AACSS).

Readers comments and suggestions are welcome. For further information or additional copies of this document, write or phone the CECOM Safety Office.

Commander
U.S. Army Communications-Electronics Command
ATTN: AMSEL-SF-SEP (Leonard Russo)
Fort Monmouth, New Jersey 07703-5024
(COM) 908-532-0084 (DSN) 992-0084
(FAX) 908-995-2667

TABLE OF CONTENTS

## 1. Introduction

In the past, industry in general considered increased productivity as the most important aspect of Software Engineering. Very little was mentioned about the reliability of the software product and nothing about the safety of the software product.

In recent years, the role of software has become integral to the command and control of complex and costly systems upon which human lives may depend. This role has compelled both the Department of Army and Industry to establish goals of highly reliable and productive safe software in which hazard-causing faults or errors are unacceptable. These new goals require the support of professionals who have attained some level of expertise in the various aspects of software and firmware. The Safety Engineer should be able to apply system safety methods and techniques to the analysis of software with a reasonably high level of confidence in order to certify the safety of the overall system.

Because Software Safety is a field in its infancy, all of the usual birthing and growing pains have to be experienced. Recent incidents involving software are strongly suggestive of the risks involved. System Safety Engineers need to recognize that software is just another system component, and that this component can contain errors or defects which can cause undesired events in the overall system. System Safety Engineers must work in conjunction with Software Engineers to identify those errors which can cause hazards or produce undesired events.

## 2. Getting a Software Safety Program Started

Most new materiel development efforts are predominantly software-controlled. The Operational Requirements Document (ORD) is the first document that should specify software and hardware hazards that will be eliminated or controlled. By specifying software hazards in the ORD, the System Safety Engineer has the basis for initiating a Software Safety program. Often, at this point in time, it is very difficult to do this, since requirements are not well defined. The System Safety Engineer should include generalized statements to cover these hazards. The System Safety lessons learned data base for embedded software may be able to give Engineers some examples.

Software Safety should then be included as part of the Request For Proposal (RFP). Inclusion of Software Safety into the RFP will alert the contractor that a Software Safety effort will be required as part of the contract. The following items can be included in the RFP:

a. The development of any firmware associated with safety-related functions, data, or storage should be controlled as software instead of hardware. This requires that firmware have increased configuration control, testing, and quality assurance (see Section 12, def. of Firmware).

b. A Hardware Risk Assessment Matrix and a Software Safety Matrix should be specified in the RFP. The combined use of these matrices will assist in identifying the required safety effort (see Section 3).

c. Examples of anticipated hazards should be provided to the contractor via the Software Safety Specification/Guidelines, that are referenced in the SOW. This information is typically available from mishap/incident reports associated with similar systems (in the field as well as the test environment). It is expected that a contractor will, at least, repeat these in his proposal. Example hazards should include software or hardware-effected failures.

d. The definition of risk in DOD-STD-2167A should be expanded to include Safety risk (see Section 12, def. of risk management).

e. The RFP should require a Software Safety Program Plan (SWSPP), using DI-SAFT-80100, to be submitted as an appendix to both the System Safety Program Plan (SSPP) and the Software Development Plan (SDP). The System Safety engineer can judge the level of integration of the Software Safety tasks, manning, and liaisons as referenced in both the SSPP and SDP (see Section 4).

f. The Request for Proposal (RFP) should also require a Software vs. Hardware Safety lifecycle flow diagram to be submitted as part of the proposal (see Figure 1).

g. A draft Software System Safety Design Verification Checklist should also be included in the RFP (see appendix A). This will enable Software and Safety Engineers to better assess and identify the safety critical requirements and interfaces. This checklist can then be tailored and used to develop software safety tests. Furthermore, it will also ensure traceability to the requirement/specification documents.

## 3. Software Safety Hazard Assessment Process

When implementing software safety as part of an overall system safety effort, it is critical to specify a Software Safety matrix as well as a hardware matrix. The hardware matrix is a familiar tool but the Software Safety matrix must be approached from a different perspective.

### 3.1 Software Hazard Criticality Matrix

The Software Hazard Criticality Matrix (See Figure 2) is similar in form to the Hazard Risk Assessment Matrix for hardware, but the purpose is to define the level of testing rather then the hazard category. The matrix is established using the Software Hazard Severity Categories for the columns and the Software Hazard Control Categories for the rows. The matrix is completed by assigning a Software Hazard Assessment Code Index number to each element. A Software Hazard Assessment Code (SHAC) of "1AT, 1AN, 1IT, 1OC, 1ID, 2AT, 2AN, or 2IT" from the matrix implies that a significant and rigorous amount of analyses/testing is needed. A SHAC of "2ID, 2OC, 3AT, 3AN, or 3OC" implies that in-depth testing

and a high level analysis of requirements and design are needed. A SHAC of "3OC, 3ID, 4AT, 4AN, 4IT, 4OC, or 4ID" implies that some level of testing is needed and requires verification from the managing activity.

**(NOTE: Unlike the hardware related RAC, a low index number does not mean that a design is unacceptable. Rather, it indicates that greater resources need to be applied to the analysis and testing of the software and its interaction with the system to reduce the system risk to an acceptable level.)**

A hazard index code should be assigned to each identified software-related hazard of the program using the Hazard Criticality Matrix.

3.2 Hardware vs. Software Safety Matrix

A System Risk Assessment Matrix and a Software Hazard Criticality Matrix should be included in the RFP. This will assist the contractor in initial Design Trade studies. Note that the Hardware Risk is a combination of severity and probability of the hazard under investigation. A ranking system of high, medium, and low should be furnished. The Software Safety Matrix is used when software may control, monitor, or contribute to a system level safety hazard. The system-level hazard severity is fixed by the system architecture and other design parameters. The row/column position of the software under investigation will determine the level of rigor to which Software Safety analyses, configuration control, software engineering, and quality assurances will be exercised to assure that the software safely performs, in the mission environment, the functions for which it was designed.

3.3 Software Hazard Control Categories

The specified levels of software control for hazardous function software are as follows:

a. **Autonomous Time Critical** - Software exercises autonomous control over potentially hazardous hardware systems, subsystems or components without the possibility of real time human intervention to preclude the occurrence of a hazard. Failure of the software or a failure to prevent an event leads directly to a hazard's occurrence. An example would be an aircraft automatic landing system.

b. **Autonomous/Not Time Critical** - Software exercises control over potentially hazardous hardware systems, subsystems, or components allowing time for human intervention by independent safety systems to mitigate the hazard. However, these systems themselves are not considered adequate for safety; therefore, corrective action may be necessary. An example would be an automatic terrain following flight control system.

c. **Information/Time Critical** - Software item displays information requiring immediate operator action to mitigate a hazard. Software failures will allow or may not be designed to prevent the hazard's occurrence. An example would be missile range control safe flight path parameters.

d. **Operator Control** - Software items issue commands over potentially hazardous hardware systems, subsystems or components requiring human action to complete the control function. There are several, redundant, independent safety measures for each hazardous event. An example would be weapon release from a traditional aircraft.

e. **Information Decision Algorithm** - Software generates information of a safety critical nature used to make safety critical decisions. There are several, redundant, independent safety measures for each hazardous event.

f. **Not Safety Related Software** - Software which controls no hazardous functions. This designation is required to:

(1) Show the limits of safety analysis and test.

(2) Document reasons for limits of effort.

(3) Get customer approval of the set limits.

(4) Conserve resources to lower cost.

4. Software Safety Program Plan (SWSPP)

The development and support of a safety critical software system requires application of specific system and software engineering techniques within a safety management framework. Safety management makes explicit the safety-related activities in each phase of the software lifecycle. The SWSPP shall describe how the organization should plan, develop, implement, and maintain

an effective Software Safety Program. The SWSPP shall specify safety-related activities to be carried out for each development activity. The SOW will give the details/contents that should be included in the SWSPP, as follows:

(1) Until the contractor's experience levels increase, and to assist in grading a proposal, the RFP should require a Software Safety Program Plan (SWSPP) (using DID-SAFT-80100), and be submitted as an appendix to both the Software Development Plan (SDP) (DI-MCCR-80030A), and the System Safety Program Plan (SSPP). The measures of compliance will be based on a credible state-of-the-art SWSPP and the level of integration of the SWSPP tasks, manning, and liaisons as referenced in the SSPP and SDP.

(2) The contractor shall develop and implement a method of identifying safety software functions and requirements in the software documentation. The safety critical functions should be designated as Safety Critical Computer Software Components (SCCSCs) or Safety Critical Computer Software Units (SCCSUs). During the software safety process, the contractor should ensure that subsequent software design documents identify the SCCSCs and SCCSUs appropriately.

(3) The contractor should develop and implement a tracking system within the configuration management structure for SCCSCs and SCCSUs and safety design requirements. The tracking system should include the flow of safety functions and requirements through the software documentation. The tracking should include a description of the requirement and references to the implementation of each requirement at each level of documentation.

(4) The contractor's System Safety Organization should participate in the development of the Software Development Plan (SDP), Software Quality Evaluation Plan (SQEP), and other documents governing the contractor's software development and evaluation process. System Safety will incorporate appropriate design guidelines and requirements. The contractor shall develop Software Safety Evaluation criteria and incorporate these criteria into the SQEP.

## 4.1 Software Development Plan

The Software Development Plan should describe:

(1) how the software requirements are derived from the initial requirements of the system,

(2) when and how the program is structured and coded,

(3) when a hazard analysis is performed on the software/system/component and whether this analysis remains ongoing throughout development as revisions and enhancements occur,

(4) when verification, validation and design reviews are conducted,

(5) what criteria must be met for a formal release of the software,

(6) what procedure are used to ensure that all software revisions and testing requirements are traceable to corresponding software or program requirements,

(7) what procedures are used to assure that software and system testing requirements are kept current with revisions, and

(8) what quality assurance procedures are followed during design and development phases.

## 4.2 Documentation Requirements

Systematic documentation control at every phase of the development cycle is an essential part of ensuring the safety of the design and implementation. Several standards (such as DOD-STD-2167A) identify documentation for both critical and non-critical software.

A section of the SWSPP shall specify documents to be used and their contents. The development organization may elect to prepare independent safety documents or may integrate the safety documentation with other program documents. Whichever form is chosen, the following additional documentation requirements exist for safety critical software.

(1) Results of Software Safety Requirements Analysis

(2) Results of Software Safety Design Analysis

(3) Results of Software Safety Code Analysis

(4) Results of Software Safety Test Analysis

For each of the reports listed above, the SWSPP should define its content.

### 4.3 Additional Documentation

Documents prepared IAW existing standards (i.e., DOD-STD-2167A) can provide the framework within which many software safety program needs can be accomplished. However, these standards do not specifically highlight safety related activities. The information specific to the Software Safety Program may be included through augmentation of the named documents. The following documents should be addressed:

(1) **Software Program Management** - Documentation of how the Software Safety Program will be implemented, integrated, and managed with other development activities. The Software Development Plan (DOD-STD-2167A), can be augmented to include this information.

(2) **Software Safety Requirements** - Specification of safety requirements to be met by the software to avoid or control safety hazards should be prepared. The Software Requirement Specification (SRS) (DOD-STD-2167A) can be augmented to include this information.

(3) **Software Development Standards, Practices, and Conventions** Approved, controlled, and/or prohibited practices that are essential to achieve system and software safety objectives and requirements should be specified.

(4) **Test Documentation** - Specific test planning, test design, test cases, test procedures, and test reports should be prepared and/or performed to demonstrate that safety requirements are satisfied. The Test Documentation identified in DOD-STD-2167A may be augmented to include this information. The final Test Report should include and assess the residual safety risk with respect to incomplete changes or updates to the software and the system level hardware. This is essential, since these documents will be used for preparing the software materiel release statement.

(5) **Software Verification and Validation** - Information regarding how Software Safety will be verified and validated should be prepared. The method(s) to ensure the traceability of safety requirements to the specifications, implementation, and safety related test cases should be specified. The formal report submitted by the independent IV&V agent (IAW DOD-STD-2167A section 4.1.7) should include the results of safety-related V&V

activities. This is in addition to the requirements IAW DOD-STD-2167A, for general software engineering practices. The results can be included or combined with other IV&V or V&V efforts. These results will also be analyzed to ensure that no residual hazards exist in order to prepare a software materiel release statement.

(6) **Software User Documentation** - Information that may be significant to the safe installation, use, maintenance, and/or retirement of the system should be prepared. The Software Users documentation described in (DOD-STD-2167A) may be augmented to include this information.

## 5. Configuration Management Activities

The correct configuration of the safety critical software is an essential element in the overall integrity of the system being developed. Rigorous configuration management must be in force during all phases of the software lifecycle, from project initiation through system retirement, and shall include appropriate control of program documentation, source code, object code, data development tools, environments (both hardware and software), and test cases. Software Safety configuration control, in some cases, should be required in the SWSPP. A section of the SWSPP should describe how the software configuration should be managed IAW an approved Configuration Management Plan (CMP). Approved methods and/or tools should be used for configuration control, access control, and status reporting. DOD-STD-2167A section 4.5 provides guidance on planning software configuration management practices. Particular attention shall be paid to the process by which changes to specific safety critical software items are authorized and access granted to specific safety critical configuration items for incorporation of approved changes.

### 5.1 Configuration Control Board

A section of the SWSPP should also include a description of the roles and responsibilities of the safety personnel in the change evaluation, change approval, and change verification processes. The relationship between the Configuration Control Board (CCB) and other boards, which may have safety related responsibilities, should be identified. It is strongly recommended that the CCB have a Software Safety representative as a member.

### 5.2 Additional Requirements

A description of the provisions for ensuring that configuration management of the following software meets the

additional requirements necessary for safety critical software should be prepared and included in the SWSPP:

(1) Software Development Tools

(2) Previously Developed Software

(3) NDI Software

(4) Subcontractor Developed Software

## 6. Previously Developed or NDI/Commercial Off-The-Shelf (COTS) Software

Previously developed or GFE software (i.e., operating systems, scientific subroutine libraries, display management systems, data base system, etc.) may be used in whole or in part to satisfy system requirements. Additionally, this software may permit or require its characteristics to be tailored or adapted through linking to other software or through access to data. The use of previously developed or purchased software in a safety critical application does not exempt that software from provisions of the SWSPP.

### 6.1 Safety Critical Approval Process

The use of previously developed or purchased software in a system with safety critical operations necessitates completing an approval process that should include the following steps:

(1) Determine the extent to which the previously developed or purchased software will be used in a safety critical system.

(2) Identify relevant documents (i.e., product specifications, design documents, usage documents, etc.) that are available to the obtaining organization, and determine their status.

(3) Determine the conformance of the previously developed or purchased software to published specifications.

(4) Identify the capabilities and limitations of the previously developed or purchased software with the program requirements.

(5) Test the safety critical features of the previously developed or purchased software independent of the program's software.

(6) Test the safety critical features of the previously developed or purchased software with the program's software.

(7) Perform a hazard assessment to determine if the use of the previously developed or purchased software will result in undertaking an acceptable level of testing even if unforeseen hazards result in a failure.

## 6.2 Safety Critical Concerns

Previously developed or purchased software that should not be used in safety critical software products are those that:

(1) Cannot be adequately tested

(2) Present significant risk of hazardous failure

(3) Become unsafe in the context of its planned use, or

(4) Represent significant adverse consequences in the event of a failure.

The inability to determine the level of risk or the consequence of failure is justification for rejecting the use of the previously developed or purchased software. Equivalent analyses, tests, and demonstrations by the vendor of the adequacy of the vendor supplied software for use in safety critical application may be accepted as satisfying the intent of the above requirements.

Previously developed or purchased software that is obtained as source code and modified for use by the project should be subject to the same Software Safety Program requirements as are applied to new software.

## 7. Software Safety Analyses

The Software Safety tasks included in MIL-STD-882B, Notice 1, will increase the costs of the system safety program. This increased level of effort in software safety analyses will also result in a greater number of hazards requiring the Program Manager's attention. However, the initial increase in overall program development costs will result in lifecycle savings due to fewer software and system failures. It should be noted that the potential benefits of Software Safety, especially for large scale systems, should justify these costs.

The Preliminary Hazard Analyses (PHA) are the first step in determining the level of Software Safety effort required. Once the preliminary hazards are identified, these can be attributed to software or hardware. The Software Safety Program Plan (SWSPP) should address how the contractor plans to tailor the analyses, and the configuration control methodology, based on those specified hazards.

Software Safety tasks, from the 300 series of MIL-STD-882B, Notice 1, should be specified to address the identified hazards. Tailoring of these tasks is crucial in keeping the cost of the program down. Software Fault Tree Analysis, Code Analysis, Petri-Net Analysis, and other methodologies are labor intensive and should be directed toward specified hazardous conditions.

## 8. General Guidelines for Designing Safety-Critical Software

The primary events in a Software Safety program include identifying the hazardous conditions controlled by software, determining the extent of control on these software functions, and ensuring that the software is tested. The following are some guidelines that should be followed when designing safety-critical software. These guidelines should be included in the Software Safety Specification, and included as part of the Procurement Data Package input.

a. The safety-related code should be isolated physically. the safety-related software should be on as few circuit cards as possible to minimize configuration control.

b. The safety-related functions should be identified within the system. This allows the software code to be traced and checked for errors.

c. The safety-related code should be single entrance-exit so that erroneous input or output does not create a hazardous condition. The analysis and test of the software will be enhanced by decreased complexity; avoid "spaghetti code."

d. The software should always initialize in a known safe state.

e. The complexity of safety-related software should be considered in the software design. Complexity can be controlled by eliminating "GO TO" and "with"ing multiple packages together in Ada, and by optimizing the number of safety interlocks to pertinent-state description input and diverse-state sample testing.

f. The environment of operation should be considered when designing the location of software within the system.

g. The software should be designed such that a single point failure cannot create a hazardous condition.

h. System and Subsystem specifications should be reviewed to identify and verify operator-software interfaces.

i. Functional flow diagrams (or their functional equivalent), storage allocation charts, and other program documentation should be analyzed to ensure specification and safety requirements have been met.

j. The software should perform a status check of safety-critical elements prior to executing a potentially hazardous sequence.

k. The software should incorporate a minimum of two separate independent commands to initiate a safety-critical function.

l. Safety Critical Computer Software Components (SCCSCs) in which changes have been made shall be subjected to complete regression testing. This is applicable to all software development projects where version updates and enhancements, to fielded and developmental software, are made in order to sustain full materiel release of the software. All changes (i.e., code level) must be analyzed, reassessed, and tested to ensure resolution. All documentation should be updated, and Configuration Control/Management of changes must include a safety sign-off.

## 9. Conclusion

Lessons learned are extremely important. Until Data Recorders or other On-Board Diagnostic Recorders are available, only corporate memory or tribal knowledge will keep us from repeating past mistakes. At some point in the near future, the Software Safety Action Committee will call for submittal (see Foreword of this document), of all available lessons learned to compile a sanitized list, which will then be made available to all users via a database. It was pointed out in the Software Safety course that lessons learned will be crucial to passable Software Safety on any

project. An appendix to this document is being established which will contain software safety lessons learned/incidents. This will be constantly updated as reviewers enter their specific software safety hazards.

## 10. Recommendations

Training for Software Safety is available from:

* McKinlay & Associates
  15669 Century Lake Dr.
  Chesterfield, MO. 63017
  Instructor: Archibald McKinlay (314) 532-2136/5657
  3 days at your location

(or)

* University of Southern California
  Institute of Safety & Systems Management
  Los Angeles, CA.
  Instructors: A. McKinlay, & Dr. Gerald McDonald
  3 days at USC

* University of Maryland
  Software Reliability Course
  Instructor: Dev Reheja
  2 days at University of Maryland

The AMC Action Committee received training from McKinlay & Associates.

Future actions of the Software Safety Action Committee will address: (1) Safety Statements in the Operational Requirements Document (ORD), (2) Software Safety Analysis Techniques, (3) Software Safety in projects already in development, and (4) testing products which have not received the benefits of Software Safety Engineering.

## 11. References


MIL-STD-882B — System Safety Program Requirements, 30 Mar 84 (change Notice 1)

AR 385-16 — System Safety Engineering and Management, 3 May 90

DOD-STD-2167A — Defense System Software Development, 29 Feb 88 (Safety related examples are in MIL-HDBK-287)

DOD-HDBK-287 — Defense System Software Development Handbook, 20 Apr 88

DOD-STD-2168 — Software Quality Evaluation, 26 Apr 85

NATO STANAG 4404 — Safety Design Requirements and Guidelines for 1990, Munitions Related Safety Critical Computing Systems.

DEF-STAN-00-55 — Requirements for the Procurement of Safety, UK Critical Software in Defense Equipment.

## 12. Definitions, Terms, and Acronyms

The following are definitions of words, terms, and acronyms used in this document, or in other documents relating to software and software safety.

(NOTE: Although the following terminology is relatively new to the safety community, it is widely accepted and used in various military and industry standards.)

Computer Hardware - Devices capable of accepting and storing computer data, executing a sequence of operations on computer data, or producing any output data (including control outputs).

Computer Software (or Software) - A combination of associated computer instructions and computer data definitions required to enable the computer hardware to perform computational or control functions.

Computer Software Component (CSC) - A distinct part of a computer software configuration item (CSCI). CSCs may be further decomposed into other CSCs and Computer Software Units (CSUs).

Computer Software Configuration Item (CSCI) - Software that is designated by the procuring agency for configuration management.

Computer Software Unit (CSU) - An element specified in the design of a Computer Software Component (CSC) that is separately testable.

Firmware - Software that resides in a nonvolatile medium that is read-only in nature, and is completely write-protected when functioning in its operational environment (i.e., PROM, ROM, EPLD, PLA, and MMIC as well as transistor circuits) Firmware for safety-related functions, data, or storage, shall be controlled as software during development and test. The contractor is responsible for the safety of commercial firmware, and this should be reflected in the contract and RFP.

Hazard - A condition that is a prerequisite for an accident. (See AR 385-16.)

Hazardous Operation/Condition - An operation (activity) or condition (state) which introduces a hazard to an existing situation without adequate control of that hazard, or removes or

reduces the effectiveness of existing controls over existing hazards, thereby increasing mishap probability or potential mishap severity, or both.

Independent Verification and Validation (IV&V) - An independent test and evaluation process that ensures that the computer program satisfactorily performs, in the mission environment, the functions for which it was designed. Verification is the iterative process of ensuring that during each phase of development the software satisfies and implements only those software safety requirements that were approved at the end of the previous phase. Validation is the test and evaluation process to ensure that the software meets all system and software safety performance requirements.

Managing Activity - The organizational element of DOD assigned acquisition management responsibility for the system; or prime or associate contractors or subcontractors who wish to impose system safety tasks on their suppliers.

Memory - An electronic, mechanical, magnetic, or other technology device, or location within such a device, where software data is stored.

Mishap Probability - The numerical likelihood that a mishap will occur given a defined set of circumstances. This term does not reflect the reliability of the software, which is the likelihood of a software component error (commonly referred to as a software "bug") rendering the software useless. Safety analyses assume a software reliability of 1. (See MIL-HDBK-217.)

Non-Safety-Critical Computer Software Component - Computer software component (unit) which does not control safety-critical hardware systems, subsystems, or components, and does not provide safety-critical information.

Product Baseline - Configuration Item(s) which have design frozen at established program milestones (SDR, PDR, CDR), and ultimately subjected to formal testing and configuration audits prior to delivery.

Reduced Instruction Set Chip (RISC) - The RISC style architecture may be said to be characterized by the following features:

(1) A small set of primitive instructions of essentially the same size is available.

(2) Each instruction is executed in one machine cycle.

(3) The instructions provide support for high level languages and their compilers.

(4) Only load/store instructions access memory. The other instructions operate upon registers.

Risk Management - The process whereby management decisions are made and implemented regarding the control of risks. The contractor shall document and implement procedures for risk management. The contractor shall identify, analyze, prioritize, and monitor the areas of the software development project that involve potential technical, cost, schedule, and safety risks.

Safety-Critical Computer Software Component (SCCSC) - Computer software component (unit) whose inadvertent response to stimuli, failure to respond when required, response out-of-sequence or in unplanned combination with others, can result in a critical or catastrophic mishap, as defined in MIL-STD-882B.

Safety Integrity - The ability of a control system to work correctly (this includes shutting down safely if a fault occurs), which depends on the entire system, not just the computer.

Safety Kernel - An independent computer program that monitors the state of the system to determine when potentially unsafe system states occur or when transitions to potentially unsafe system states may occur. The Safety Kernel is designed to prevent the system from entering the unsafe state and return it to a known safe state.

Software - All instructions, logic, and data, regardless of the medium on which it is stored, that is processed or produced by automatic machines and which is used to control or program those machines. Software, as used in the context of this document, also includes firmware and documentation associated with all of the above.

Software Error - a mistake in engineering, requirements, specifications, coding, or design of software.

Software Failure - the result of a fault or a system that doesn't meet specification.

Software Fault - the manifestation of an error.

Software System Safety - The optimization of system safety in the design, development, use, and maintenance of software systems and their integration with safety-critical hardware systems in an operational environment.

Software System Safety Analysis - The use of iterative analytic, inspection, and test techniques to optimize system safety in the design, development, use and maintenance of safety-critical computer software components of hardware systems; and, to assess residual software-related mishap risk in the system.

Support Software - All software used to aid the development, testing, and support of applications, systems, test and maintenance software. Support software includes, but is not limited to:

a. Compilers, assemblers, linkage editors, libraries and loaders required to generate machine code and combine hierarchical components into executable computer programs.

b. Debugging software.

c. Stimulation and simulation software.

d. Data extraction and data reduction software.

e. Software used for management control, software configuration management, or documentation generation and control during development.

f. Test software used in software development.

g. Design aids, such as program design language tools, and problem statement analysis tools.

h. Test and maintenance software to assist in fault diagnosis and isolation, operational readiness verification, and system alignment checkout of the system or its components. It may be used to check out and certify equipment and total system at installation, reinstallation, or after maintenance. It is also used in accordance with prescribed procedures to maintain the system throughout its operational life. It should be noted that the test and maintenance software may reside in another system such as a Test Program Set (TPS).

System Safety - The application of engineering and management principles, criteria, and techniques to optimize safety within the constraints of operational effectiveness, time, and cost throughout all phases of the system life cycle.

System Software - The totality of operational software resident in a computer (operating system, executive programs, application programs and data bases) associated with a system.

Version - An identified and documented body of software. Modifications to a version of software (resulting in a new version) require configuration management actions by either the contractor, the contracting agency, or both.

# APPENDIX A

# SYSTEM SOFTWARE SAFETY CHECKLIST

| ITEM | SYSTEM SOFTWARE SAFETY CHECKLIST | NON-APPLICABLE----- NON-COMPLIANT------ COMPLIANT---------- | ---------- ----- | ---------- | ---- |
|---|---|---|---|---|---|
| | **GENERAL/MISCELLANEOUS** | | | | |
| | Provides for precluding dependence on administrative procedures. | | | | |
| | Provides for using information control for deriving the authorization code for the activation of the authorization device. | | | | |
| | Provides that the software contains only features or capabilities required by the system, and that it does not contain additional capabilities, e.g., testing, troubleshooting, etc. | | | | |
| | Provides for positive control of system safety-critical functions at all times. | | | | |
| | Provides for safety-critical sub-routines and sub-programs to include "come from" checks to verify that they are being called from a valid calling program. | | | | |
| | **SEPARATION OF COMMANDS/FUNCTIONS/FILES/PORTS** | | | | |
| | Provides for using separate launch authorization and separate launch control functions to initiate a missile launch. | | | | |
| | Precludes the ground ordnance enabling arming code from being the same as the launch authorization code. | | | | |
| | Provides for requiring separate "arm" and "fire" commands for ordnance initiation. | | | | |
| | Precludes using input/output ports for both critical and non-critical functions. | | | | |
| | Provides for sufficient difference in addresses for critical input/output ports vs. non-critical ports that a single address bit failure does not allow access to critical functions or ports. | | | | |
| | Provides for having files that are unique and have a single purpose. | | | | |
| | **INTERRUPTS** | | | | |
| | Provides for defining specific interrupt priorities and responses. | | | | |
| | Provides for software system management of interrupt control so as to not compromise safety-critical operations. | | | | |

| ITEM | SYSTEM SOFTWARE SAFETY CHECKLIST<br>NON-APPLICABLE-----<br>NON-COMPLIANT------<br>COMPLIANT---------- | ---------- ----- | ---------- | ---- |
|---|---|---|---|---|
| | **SHUTDOWN/RECOVERY/SAFING** | | | |
| | Provides for fail safe recovery from inadvertent instruction jumps. | | | |
| | Shutdown provisions are included in software upon detection of unsafe conditions. | | | |
| | Provides for the system reverting to a known predictable safe state upon detection of an anomaly. | | | |
| | Provides for software safing of critical hardware items. | | | |
| | Provides for an orderly system shutdown as a result of a command shutdown, power interruptions, or other failures. | | | |
| | Requires that the software be capable of discriminating between valid and invalid external interrupts and shall recover to a safe state in the event of an erroneous external interrupt. | | | |
| | Provides for entry into a safe state in the event of erroneous entry into a critical routine. | | | |
| | Protects against out-of-sequence transmission of safety-critical function messages by detecting any deviation from the normal sequence of transmission. When this condition is detected, the software terminates all transmissions, recycles to a known safe state, and displays the existing status so the operator can take compensatory action. | | | |
| | Provides for initializing all unused memory locations to a pattern, that if executed as an instruction, will cause the system to revert to a known safe state. | | | |
| | Provides for identifying safing scenarios for safety-critical hardware and including them into the design logic. | | | |
| | Provides for the capability of reversing or terminating launch authorization and ordnance arming functions. | | | |

| ITEM | SYSTEM SOFTWARE SAFETY CHECKLIST<br>NON-APPLICABLE-----<br>NON-COMPLIANT------<br>COMPLIANT---------- | ---------- ----- | ---------- | ---- |
|---|---|---|---|---|
| | **PREVENTING/PRECLUDING/DISALLOWING ACTIONS** | | | |
| | Provides for preventing inadvertent generation of critical commands. | | | |
| | Provides for disallowing co-existence of potentially hazardous routines. | | | |
| | Provides for preventing bypass of safety devices during test. | | | |

| | | | | |
|---|---|---|---|---|
| | Following computer memory loading, automatic control is prevented until all data is loaded and verified. | | | |
| | Precludes inadvertent operation of data entry control to critical routines. | | | |
| | Provides for precluding a change in state if data synchronization is lost. | | | |
| | Provides for prevention of a hardware failure of power interruption from causing a memory change. | | | |
| | Provides for prevention of memory alteration or degradation over time during use. | | | |
| | Provides for program protection against unauthorized changes. | | | |
| | Provides for not allowing the safety-critical time limits in decision logic to be changed by the console operator. | | | |
| | Provides for preventing inadvertent entry into a critical routine. | | | |
| | Provides for not allowing a hazardous sequence to be initiated by a single keyboard entry. | | | |
| | Prohibits transmission of any critical command found to be in error and notifies the operator of the error. | | | |
| | Provides the controlling or monitoring of nuclear weapons to be incapable of bypassing operator control of safety-critical functions. | | | |
| | Provides for disallowing use of workaround procedures when reverting to a safe configuration after the detection of an anomaly. | | | |

| **ITEM** | **SYSTEM SOFTWARE SAFETY CHECKLIST** **NON-APPLICABLE----- NON-COMPLIANT------ COMPLIANT----------** | ---------- ----- | ---------- | ---- |
|---|---|---|---|---|
| | Provides for not using a "stop" or "halt" instruction or causing "wait" state. The CPU is always executing, whether idling with nothing to do or actively processing. | | | |
| | Provides for detection and termination of commands requesting actions beyond the performance capability of the system. | | | |
| | Provides for disallowing performance of a potentially hazardous routine concurrently with a maintenance action. | | | |
| | **MEMORY/STORAGE DATA TRANSFER** | | | |
| | Precludes storage, in usable form of information required to cause initiation of a safety-critical function. | | | |
| | Provides for self-test capability to assure memory integrity. | | | |
| | Provides for prevention of a hardware failure or power interruption from causing a memory change. | | | |
| | Provides for prevention of memory alteration or degradation over time during use. | | | |

| | | | | |
|---|---|---|---|---|
| | Provides for erasure or obliteration of clear text secure codes from memory. | | | |
| | Provides for limiting control access to storage devices memory. | | | |
| | Provides for protecting the accessibility of memory regions dedicated to critical functions. | | | |
| | Provides for having safety-critical operational software instructions resident only in non-volatile ROM. | | | |
| | Provides for not using scratch files for storing or transferring safety-critical information between computers. | | | |
| | Provides that remote transfer of data cannot be accomplished until verification of data to be transferred is accomplished and authorization to transfer the data has been provided by the operator(s). | | | |
| | Provides for self-test capability to assure memory integrity. | | | |

| ITEM | SYSTEM SOFTWARE SAFETY CHECKLIST NON-APPLICABLE----- NON-COMPLIANT------ COMPLIANT---------- | ---------- ----- | ---------- | ---- |
|---|---|---|---|---|
| | **VERIFICATION/VALIDATION CHECKS** | | | |
| | When a test specifies for the removal of safety interlocks, the software provides for verification of reinstatement of these safety interlocks at the completion of the testing. | | | |
| | Provides for verification and validation of status flags. | | | |
| | Provides for software validation of critical commands. | | | |
| | Provides for verification of the existence of prerequisite conditions prior to command issuance IAW predefined operational requirements. | | | |
| | Requires that critical data communicated from one CPU to another be verified prior to operational use. | | | |
| | Provides for verification of the results of safety-critical algorithms prior to use. | | | |
| | Provides for verification of safety-critical parameters or variables before an output is allowed. | | | |
| | Decisioning verifies the sequence and logic of all safety-critical command messages and rejects commands when sequence or logic is incorrect. | | | |
| | Provides that remote transfer of data cannot be accomplished until verification of data to be transferred is accomplished and authorization to transfer the data has been provided by the operator(s). | | | |
| | Provides that all operator actions that set up safety-critical signals are verified by software based on control device positions. | | | |
| | Provides for control of analog functions having positive feedback mecha- | | | |

| | | | | |
|---|---|---|---|---|
| | nisms that provide positive indications of the function having occurred | | | |
| | Provides for verification and validation of the prompt for the initialization of the hazardous operation or sequence of hazardous operations. | | | |
| | Provides for verification of accomplishment of each step of a hazardous operation, or sequence of hazardous operations, by setting of a dedicated status flag prior to proceeding to and initiating the next step in the operation of a series of operations. | | | |
| | Provides for verification/validation of all critical commands prior to transmission. | | | |

| ITEM | SYSTEM SOFTWARE SAFETY CHECKLIST NON-APPLICABLE----- NON-COMPLIANT------ COMPLIANT---------- | ---------- ----- | ---------- | ---- |
|---|---|---|---|---|
| | **LOGIC STRUCTURE/UNIQUE CODES/INTERLOCKS** | | | |
| | Provides for identification of flags to be unique and single purpose. | | | |
| | Provides for using unique arming codes to control critical safety devices. | | | |
| | Provides for inclusion of system interlocks. | | | |
| | Provides for using a minimum of two separate independent commands to initiate a safety-critical function. | | | |
| | Provides for the majority of safety-critical decisions and algorithms to be contained within a single (or few) software development module(s). | | | |
| | Provides for single CPU control of a process which can result in major system loss, system damage, or loss of human life to be incapable of satisfying all of the requirements for initiation of the process. | | | |
| | Requires that decision logic using registers which obtain values from end-item hardware and software not be based on values of all "ones" or all "zeroes." | | | |
| | Requires that decision logic using registers which obtain values from end-item hardware and software use specific binary patterns to reduce the likelihood of malfunctioning end-item hardware/software satisfying the decision logic. | | | |
| | Provides for cooperative processing between launch control point and mis-sile computer(s) to process safety-critical functions. | | | |
| | Provides for having safety-critical modules with only one entry and one exit point. | | | |
| | Provides for having files that are unique and have a single purpose. | | | |
| | Provides for not having operational program loads contain unused executable code. | | | |
| | **REASONABLENESS CHECKS** | | | |
| | Provides for software system reasonableness checks on all safety-critical inputs. | | | |
| | Provides for performing parity or other checks requiring two decisions before providing an output. | | | |

| ITEM | SYSTEM SOFTWARE SAFETY CHECKLIST<br>NON-APPLICABLE-----<br>NON-COMPLIANT------<br>COMPLIANT---------- | ---------- ----- | ---------- | ---- |
|---|---|---|---|---|
| | **MONITORING/DETECTION** | | | |
| | Provides for inclusion of monitoring of safety devices. | | | |
| | Provides for detection or inadvertent computer character outputs. | | | |
| | Provides for detection of errors during computer memory loading to terminal loading process. | | | |
| | provides for detection of unauthorized operation of data entry control. | | | |
| | Provides for identification of safety-critical functions requiring continuous monitoring. | | | |
| | Provides for detection of improper processing that could degrade safety. | | | |
| | Provides for detecting a predefined safety-critical anomaly and informing the operator what action was taken. | | | |
| | Requires that the software be capable of discriminating between valid and invalid external interrupts and shall recover to a safe state in the event of an erroneous external interrupt. | | | |
| | Provides for detection of improper sequence requests by the operator. | | | |
| | Provides for detection of inadvertent transfer of safety-critical routines. | | | |
| | Provides for detection and termination of commands requesting actions beyond the performance capability of the system. | | | |

| ITEM | SYSTEM SOFTWARE SAFETY CHECKLIST<br>NON-APPLICABLE-----<br>NON-COMPLIANT------<br>COMPLIANT---------- | ---------- ----- | ---------- | ---- |
|---|---|---|---|---|

| | | | | |
|---|---|---|---|---|
| | **INITIALIZATION/TIMING/SEQUENCING/STATUS CHECKING** | | | |
| | Provides for a status check of critical system elements prior to executing a potentially hazardous sequence. | | | |
| | Provides the proper configuration of inhibits, interlocks, safing logic, and exception limits at initialization. | | | |
| | Provides for issuance of good guidance signal subsequent to satisfaction of performance of flight safety checks. | | | |
| | Provides for timing sufficiency of commands relative to response to detected unsafe conditions. | | | |
| | Provides for software initialization to a known safe state. | | | |
| | Provides that all critical timing relative to hazardous operations processing is automated. | | | |
| | Provides for employing time limits for operations impacting system safety and having these time limits included in the decision logic. | | | |
| | Provides for initializing all unused memory locations to a pattern, that if executed as an instruction, will cause the system to revert to a known safe state. | | | |
| | Provides for matching of observed flight terrain being matched to computer stored flight terrain map prior to issuance of critical commands (e.g., arming, fire, climb descend, etc.) | | | |
| | Applies the use of software timing coincident with hardware timing to prevent initiation of safety-critical functions. | | | |

| **ITEM** | **SYSTEM SOFTWARE SAFETY CHECKLIST** **NON-APPLICABLE----- NON-COMPLIANT------ COMPLIANT----------** | ---------- ----- | ---------- | ---- |
|---|---|---|---|---|
| | **OPERATOR RESPONSE/LIMITATIONS** | | | |
| | Requires an operator response for initiation of any potentially hazardous sequence. | | | |
| | Provides for not allowing the safety-critical time limits in decision logic to be changed by the console operator. | | | |
| | Provides for concise definition of operator interactions with the software. | | | |

| | | | | |
|---|---|---|---|---|
| | Provides for the operator cancellation of current processing in a safe manner. | | | |
| | Requires that an operator cancellation of current processing be verified by an additional operator response. | | | |
| | Provides that the system responds to a predefined safety-critical anomalous conditions by notifying the operator of the condition and identifying the action taken. | | | |
| | Provides that upon safing the system, the resulting system configuration or status be provided to the operator and await definition of subsequent software activity. | | | |
| | Provides that remote transfer of data can not be accomplished until verification of data to be transferred is accomplished and authorization to transfer the data has been provided by the operator. | | | |
| | Provides that operator control of safety-critical functions is maintained under all circumstances throughout the weapon system operation. | | | |
| | Provides that all manual actions that set up safety-critical signals are verified by software based on control device positions. | | | |

| ITEM | SYSTEM SOFTWARE SAFETY CHECKLIST NON-APPLICABLE----- NON-COMPLIANT------ COMPLIANT---------- | ---------- ----- | ---------- | ---- |
|---|---|---|---|---|
| | **OPERATOR NOTIFICATION** | | | |
| | Requires that an override of a safety interlock be identified to the test conductor by a display on the test conductor's panel. | | | |
| | Provides for generation of critical status to operator. | | | |
| | Provides to operator identification of overrides to safety interlocks. | | | |
| | Provides for software indication if unauthorized action has taken place. | | | |
| | Provides for the system informing the operator of the anomaly detected. | | | |
| | Provides the system configuration status to operator upon safing of safety-critical hardware items. | | | |
| | Provides for positive reporting of changes of safety-critical states (e.g., absence of an armed indication does not constitute a safe condition). | | | |
| | Provides for detecting a predefined safety-critical anomaly and informing the operator what action was taken. | | | |
| | Provides for the software system to display safety-critical timing data to the operator. | | | |

| | | | | |
|---|---|---|---|---|
| | Provides for the software systems to indicate to the operator the currently active operation(s) and function(s). | | | |
| | Provides for identification to the operator that a safing function execution has occurred: provides the reason for the execution with a description of the safing action. | | | |
| | Provides for notification of improper keyboard entries by the operators. | | | |
| | Prohibits transmission of any critical command found to be in error and notifies the operator of the error. | | | |
| | Provides that upon safing the system, the resulting system configuration or status be provided to the operator and await definition of subsequent software activity. | | | |

## DISTRIBUTION LIST

McKinlay & Associates
ATTN:  (Archibald McKinlay, VI)
15669 Century Lake Dr.,
Chesterfield, MO  63017

Commander
US Army Defense Ammunition Center and School
ATTN:  SMCAC-SF (Kenneth Proper)
Savanna, IL  61074-9639

Commander
US Army Aviation Systems Command
ATTN:  AMSAV-XAP (Howard Chilton)
4300 Goodfellow Boulevard
St. Louis, MO  63120-1798

Commander
US Army Missile Command
ATTN:  AMSMI-SF-SS (William Pottratz, Terrell Swindall)
Redstone Arsenal, AL  35898-5130

Commander

US Army Materiel Command
ATTN: AMCSF-E (Susan Jervis)
5001 Eisenhower Ave.
Alexandria, VA 22333-0001

Commander
US Army Safety Center
ATTN: CSSC-SE (Russ Peusch)
Fort Rucker, AL 36362-5363

Commander
US Army Test and Evaluation Command
ATTN: AMSTE-ST-S (Peter Kamenik)
Aberdeen Proving Ground, MD 21005-5059

Commander
US Army Chemical Research, Development
and Engineering Center
ATTN: SMCRC-CMS (Peter Speath)
Aberdeen Proving Ground, MD 21010-5423

CONTINUATION OF DISTRIBUTION LIST

Commander

US Army Combat Systems Test Activity
ATTN:  STECS-SO (Rudolfo Gil)
Aberdeen Proving Ground, MD  21010-5059

Commander
US Army Belvoir Research, Development
  and Engineering Center
ATTN:  STRBE-TQS (Gaines Ho)
Fort Belvoir, VA  22060-5606

Project Manager for Training Devices (PM-TRADE)
ATTN:  AMCPM-TND-SP (Connie Perry)
Naval Training Systems Center
12350 Research Parkway
Orlando, FL  32826-3224

Commander
US Army Training and Doctrine Command
ATTN:  ATOS (Gregory Skaff)
Fort Monroe, VA  23651-5000

Commander
US Army Strategic Defense Command
ATTN:  CSSD-SO (Russ Hutcherson)
PO Box 1500
Huntsville, AL  35807-3801

Administrator
Defense Technical Information Center
ATTN:  DTIC FDAC
Cameron Station
Alexandria, VA  22304-6145  (2 copies)

Director
US Army Materiel Systems Analysis Activity
ATTN:  DRXSY-MP
Aberdeen Proving Ground, MD  21005